I2C 通信型　10 桁 7seg 表示モジュール
(カソードコモン1文字タイプ×１０個)



１．注意

本品には、7seg×10 個、制限抵抗、ピンヘッダは付属しません。

別途ご用意下さい。

２．特徴

① 7seg×10 桁の縦横幅内に回路がスッキリ収まるよう、薄型＆コンパクトに設計されています。

② 高速ダイナミック点灯により、極めてちらつきの少ない表示ができます。

③ 数値およびピリオドを含む8セグメント×１０桁分＝計８０個の LED を個別に ON/OFF 制御できます。

④ 10 桁毎に、表示輝度を１６段階で設定できます。

⑤ 7bit / 400Kbps の高速 I2C シリアル通信に対応しています。

⑥ 電源線（+5v、GND）と I2C 通信線（SCL、SDA）のたった4本だけでモジュールを動かせます。

⑦ I2C の最大の特徴である、複数のモジュールを並列に接続し、個別に制御できます。

⑧ 基板上に存在する半田ジャンパーのショートの組み合わせにより、8つのプリセットされた I2C スレーブアドレスを選ぶことができるだけでなく、プリセット以外の好きなアドレスを自由に設定できる機能もあります。もちろん設定したアドレスは内蔵のフラッシュメモリに記憶されますので、電源を抜いてもアドレスは保持されます。（尚、I2C には規格上予約された使用できないアドレスもありますのでご注意下さい）

⑨ I2C 通信線のプルアップ抵抗をモジュール上に組み込むことができます。（一般的なリードタイプの 1/4W カーボン抵抗を実装できます）

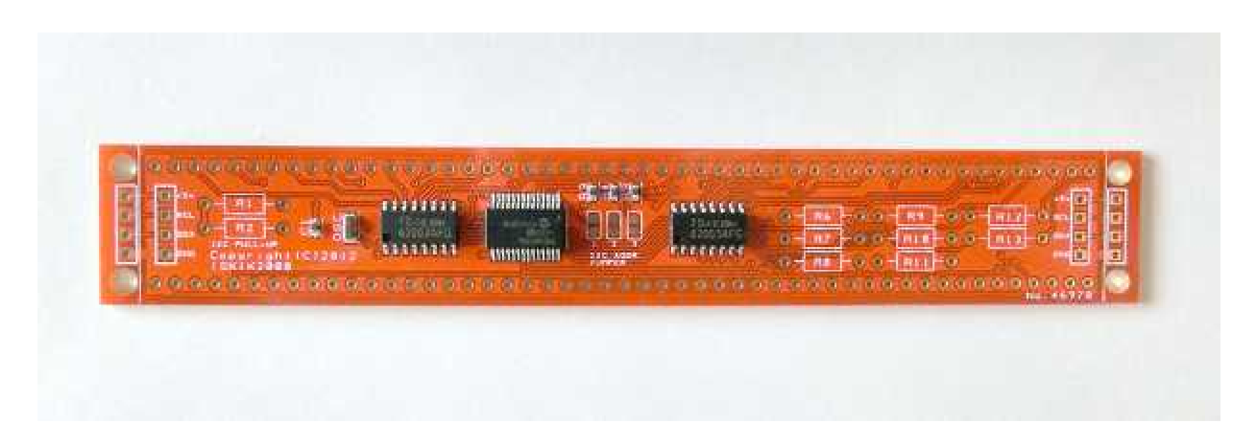
⑩ 電源線、及び通信線は、基板の左右どちらからでも取ることができます。

⑪ 接続端子は縦一列4個なので、ブレッドボードでも大変扱いやすいです。

⑫ 7seg および制限抵抗は実装されていませんので、好きな色の 7seg を実装できます。もちろん全桁を同一色にする必要はなく、たとえば赤と青を5桁ずつ実装するなど自由に設計して構いません。（順電圧、順電流の低い方に合わせた制限抵抗の考慮が必要です）　輝度に違いがある場合は輝度制御で補正することもできます。場合によっては１０桁全て付ける必要は無く、必要な桁だけ付けても構いません。

⑬ 金メッキの赤基板を採用。

３．内容



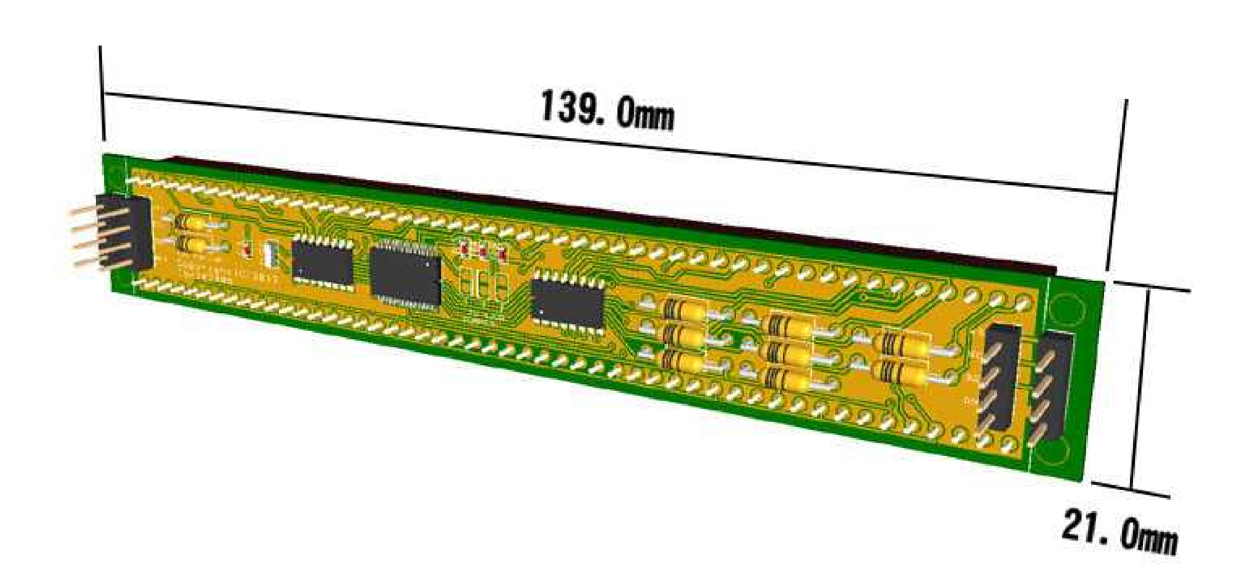
※　写真のものが全てです。 7seg、及び制限抵抗は付属しませんので、別途ご用意下さい。

※　必要に応じてピンヘッダ等ご用意下さい。

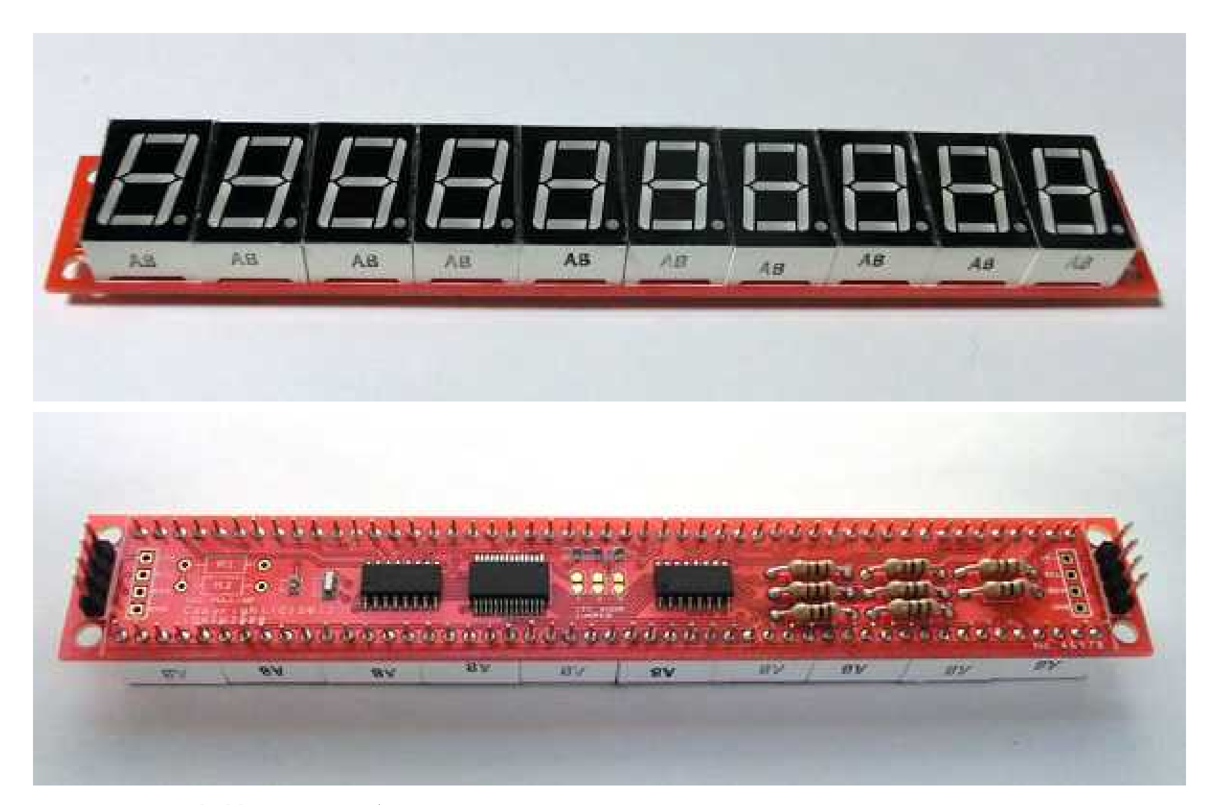
※　制御 IC、コンデンサなどのチップ部品は実装済みです。

別途ご用意頂く 7seg×10 個と、制限抵抗×8本をハンダ付けするだけで、すぐにご利用できます。

４．寸法

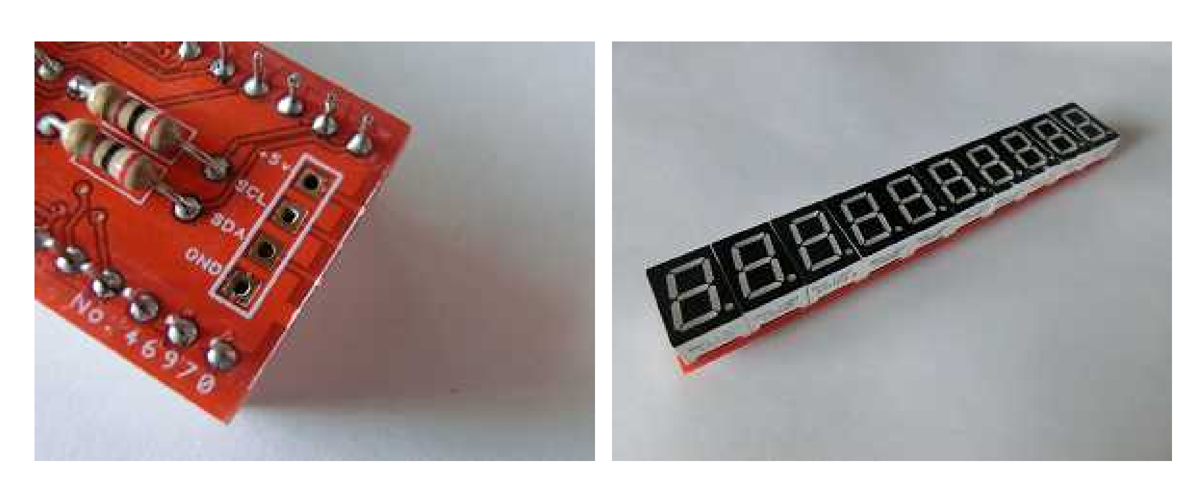


５．完成例



※ 7seg、制限抵抗、ピンヘッダは付属しません。

６．完成例　その２　（カット）

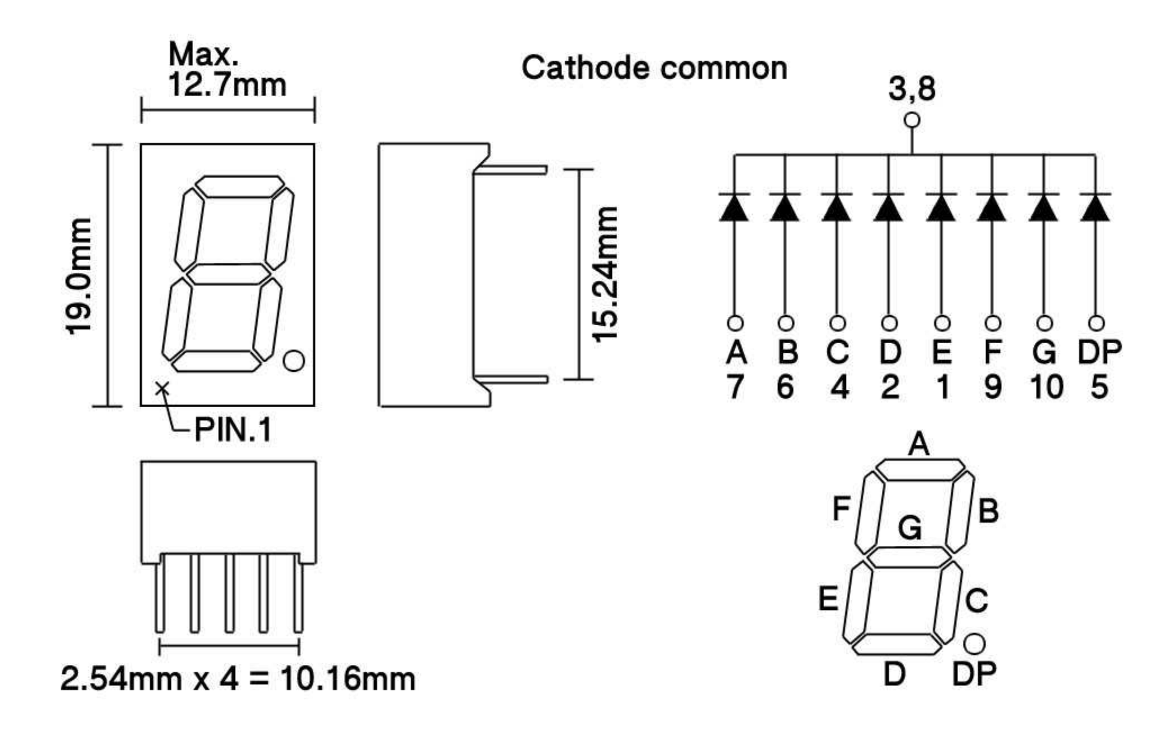


モジュール基板の両端の白い線に沿って切断することで、更にコンパクトにできます。

（プリント基板のパターンを傷つけなければ、更に 7seg サイズすれすれまで削り落とすこともできます）

尚、切断する際はアクリルカッターやヤスリ等をご利用下さい。ハサミ類では基板が割れます。

7．実装可能な 7seg の仕様



上図のような仕様のカソードコモンの1桁タイプがご利用できます。（2桁タイプは1桁タイプとピンアサインが全く異なりますのでご利用できません） リードは 2.54mm ピッチで、縦幅が 15.24mm（0.6inch）のものをご使用下さい。また、7seg 本体は、横幅が 12.7mm 以内のものをご使用下さい。 本体縦幅は 19.0mm でなくても構いません。

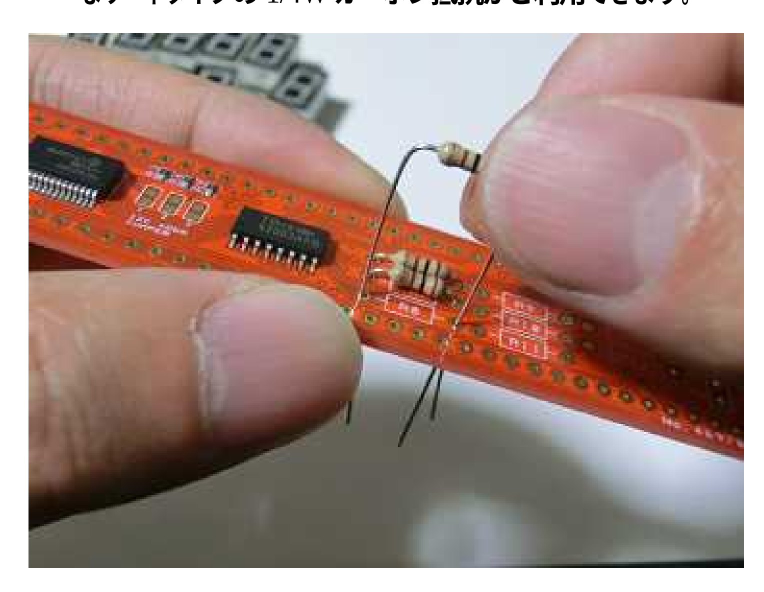


ほとんどの電子パーツショップが取り扱っている一般的な7seg ですので、容易に入手できます。

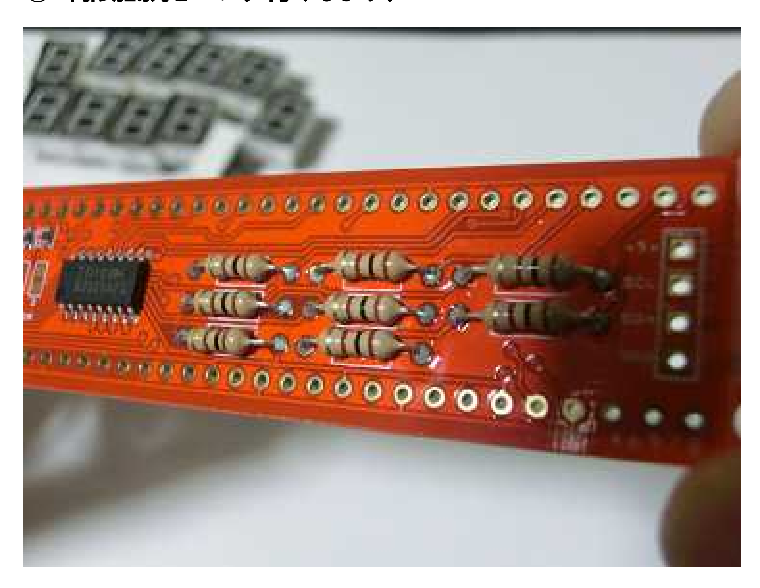
たとえば、PARA LIGHT ELECTRONICS 社製の「C-551SRD」や、OptoSupply Limited 社製の「OSL10561-LB」などがご利用できます。 製品の仕様が上記と合っていれば問題ありません。

８．制限抵抗、及び 7seg の実装手順

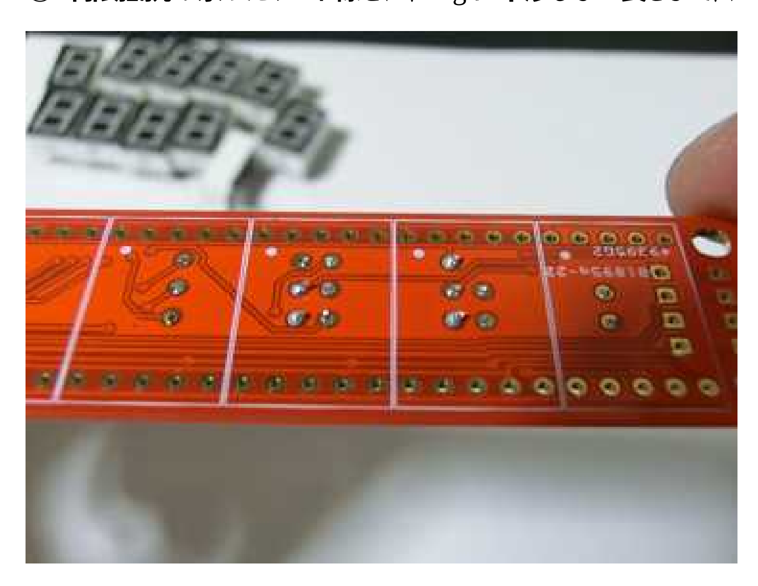
① 7seg には+5V が流れます。使用する 7seg に応じた LED の制限抵抗を 8 本（8 セグメント分）用意し、基板モジュールに挿入して下さい。基板上の挿入位置のコードは、R6〜R13 です。 抵抗器は一般的なリードタイプの 1/4W カーボン抵抗がご利用できます。



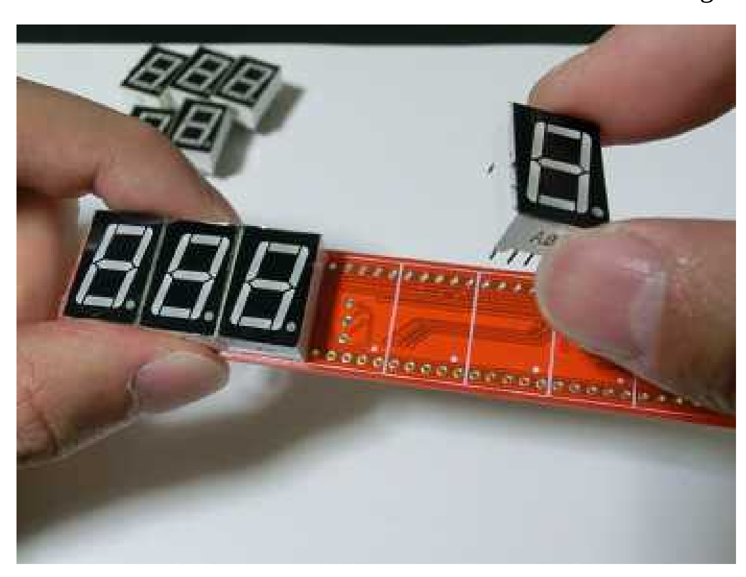
② 制限抵抗をハンダ付けします。



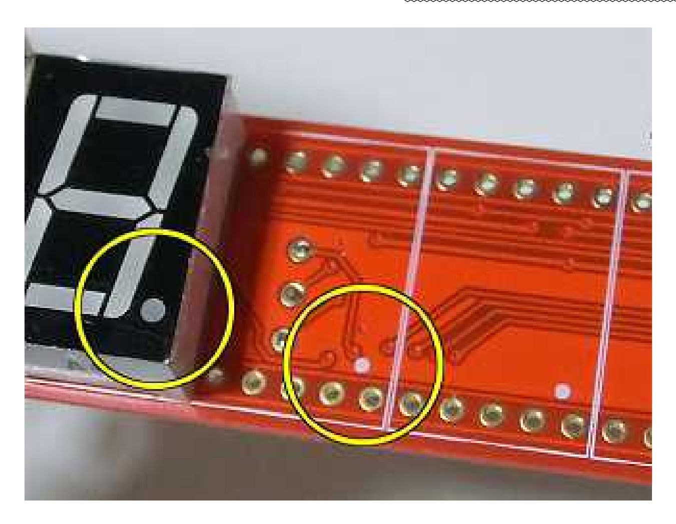
③ 制限抵抗の余ったリード線を、(7seg に干渉しない長さまで)、ニッパーで切断して下さい。



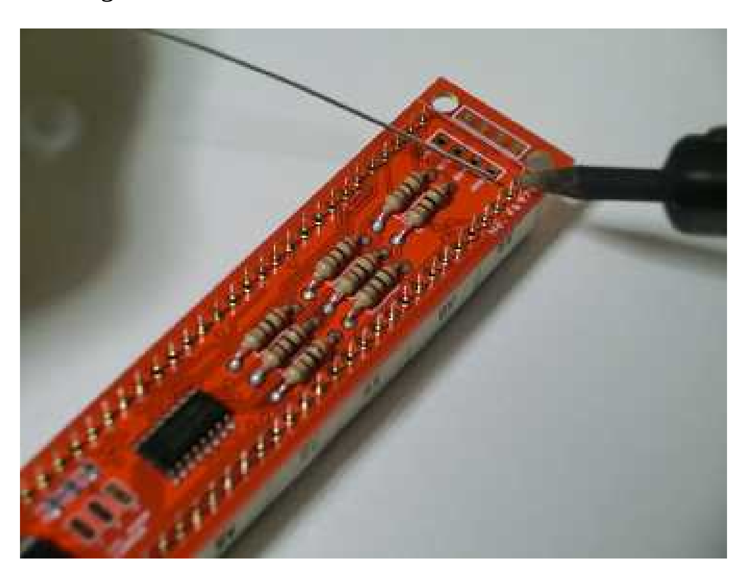
④ モジュール基板の裏面のシルクスクリーンに合わせて 7seg を乗せます。



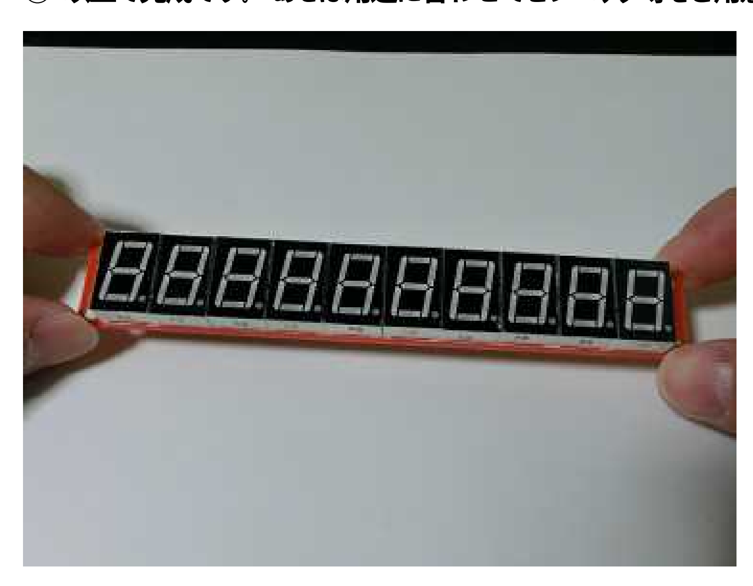
※このとき、上下を間違えないように十分にご注意下さい！！　上下はシルクスクリーンで印刷されたピリオドの位置で判断して下さい。（一度ハンダ付けしてしまうと外すのは極めて困難です！）



⑤ 7seg をハンダ付けします。



⑥ 以上で完成です。 あとは用途に合わせてピンヘッダ等をご用意下さい。

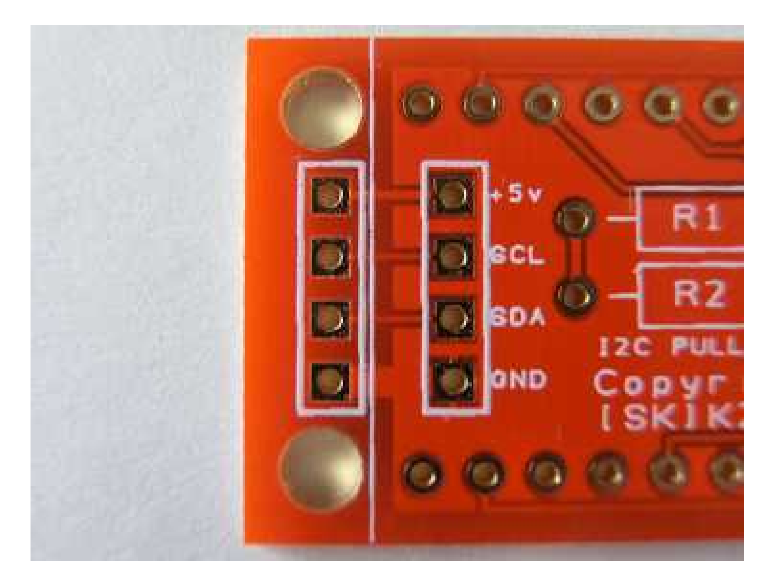


⑦ 最後に、念のために +5v と GND に DC5V (±10%) の電源を接続してください。

電源の接続時に行われる全灯点灯試験で、7seg がぼわっと点灯・消灯したら正常です。

※点灯しない場合は、DC5V の電源が来ているか、電源の接続位置は正しいか、制限抵抗は実装されているか、7seg はカソードタイプか、7seg の向きは正しいかなどをご確認下さい。

９．使用方法



モジュール基板の両側に、+5v、SCL、SDA、GND の4つの端子が存在します。 接続に必要なのはこの4端子だけです。 全部で4カ所同様の端子を設けていますが、どこを使用しても構いません。

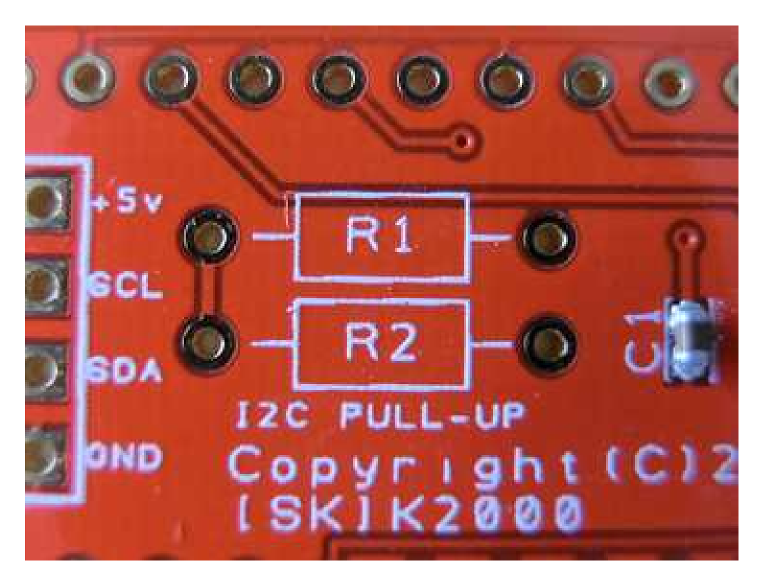
また、上図のように外側と内側にそれぞれ端子が存在しますが、内側は主に上図中央に見える白線部分を切断してコンパクト化した際に使用する端子です。

+5v には DC5V（±10%）の安定電源を。 SCL、SDA は I2C バスに接続して下さい。

※絶対に 5.5V 以上の電圧をかけないで下さい。

１０．I2C のプルアップについて

I2C バスはオープンドレイン出力のため、バスのどこかにプルアップ抵抗を入れ Vdd に接続する必要があります。 このモジュール基板にはプルアップ抵抗の接続端子を設けていますので、モジュールを単独で使用する際などにご利用下さい。 SCL、SDA の2線分の抵抗器が必要です。 抵抗値は、本来実装する回路用に計算が必要ですが、おおよそ 3.3kΩ程度のもので良いと思います。



１１．当モジュールのI2C通信の流れ

① START Condition を送信。

② 7bit の I2C スレーブアドレスを送信。

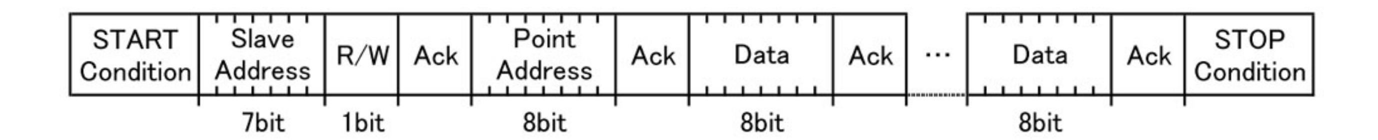
③ 1bit の Read/Write 命令を送信。(Write は 0 なので 0 を送信)

④ Ack 処理。

⑤ 8bit のポイントアドレスを送信し、Ack 処理。

⑥ 8bit のデータを送信し、Ack 処理。 (以降、必要なだけ⑥を繰り返し)

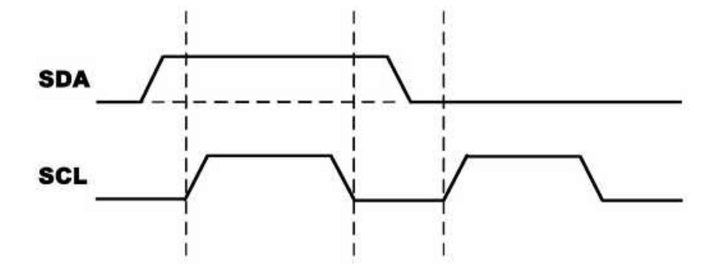
⑦ STOP Condition を送信。



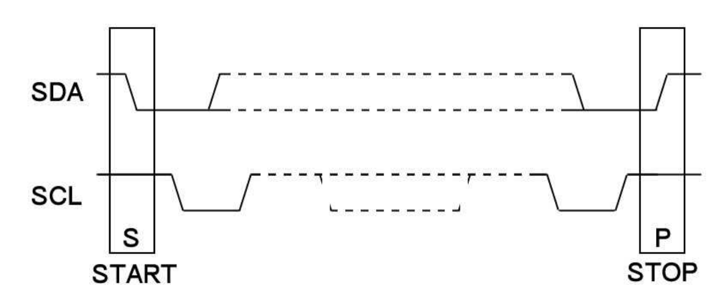
当モジュールは Read を使用しません。Write のみ使用しますので、R/W は必ず「0」になります。 R/W は Slave Address と組み合わせて8bit (1byte) として扱います。

I2C はデータの同期受信確認用にスレーブ (当モジュール) からアクノリッヂ (Ack) 情報が返されます。アクノリッヂは Lo アクティブですので、マスター側は SDA を Hi にして待ちます。送出した8bit の情報をレシーブが正しく受信すると SDA はスレーブにより Lo にされますので次のデータの送信を始めます。

I2C のデータ送出 (1bit 送出) の基本は、SDA (データライン) を Hi(1)または Lo(0)にしてから SCL (クロックライン) を Hi にして、再び SCL を Lo にしてから次のビットを SDA に指示します。 簡単なプロトコルなので、マイコンがハードウェア的に I2C マスターに対応していなくても、ソフトウェアだけで容易に実装できる点が特徴です。



START Condition、及び STOP Condition は、I2C 通信の開始と終わりを示すものです。 データ送出の場合は SCL が Lo の時に SDA を変更しましたが、START / STOP Condition の場合は、SCL が Hi の状態で SDA を Hi→Lo にすると START Condition、逆に Lo→Hi にすると STOP Condition になります。



※I2C は同期通信ですが、レシーブの性能を超える速度で通信をしようとすると、当然正しく通信はできませんので、各々要所に若干のウェイト入れが必要です。 ウェイト値は「17．デモンストレーションのサンプル回路・プログラム」に掲載されたサンプルコードを参考にして下さい。

１２．当モジュールの I2C ポイントアドレステーブル

Point Address	Description
0x00〜09	各桁の、セグメント表示設定指示。 ※0x09 を書き込むと連投状態が止まります。 （詳しくは13．セグメント表示設定指示を参照）
0x10〜14	2桁毎の HEX データ出力指示。 ※0x14 を書き込むと連投状態が止まります。 （詳しくは14．HEX データ出力指示を参照）
0x20〜29	各桁の、輝度更新指示。（設定できる値は 0x00〜0x0F の 16 段階） ※0x29 を書き込むと連投状態が止まります。 （詳しくは15．輝度更新指示を参照）
0xF0	I2C カスタムスレーブアドレス更新 （詳しくは16．I2C カスタムスレーブアドレスを参照）
0xF2	点灯試験モード設定 （詳しくは17．点灯試験モード設定を参照）

１３．セグメント表示設定指示　（Point Address = 0x00 – 0x09）

7seg の各セグメントの表示･非表示を設定するエリアです。

表示桁の左から 0x00～0x09 まで１０桁分設定でき、設定した瞬間に表示も変わります。 そのため必ずしも 0x00～0x09 の全てを更新する必要は無く、更新したい桁のポイントアドレスを直接指定して（たとえば6桁目を更新したければポイントアドレスに 0x06 を指定）、1桁だけ更新して終了しても構いません。

続けて複数桁を更新したい場合は、データを出力した瞬間にポイントアドレスも自動的にインクリメントされますので、そのまま次々とデータのみを出力するだけで更新できます。（0x09 を更新した時点で自動インクリメントはストップし、以降送信されたデータは無視されます）

出力するデータは、1バイト＝1桁分で、ピリオドを含む全8セグメントと、1バイト（8bit）は1：1で関連付いています。 各セグメントは以下のビットと対応しています。



　　H G F E D C B A
0b 0 0 0 0 0 0 0 0

つまり、「1」を表示したい場合は 0b00000110（0x06）を、「5」を表示したい場合は 0b01101101（0x6D）を送信すれば表示されます。

１４．HEX データ出力指示　（Point Address = 0x10 – 0x14）

出力したバイトデータを HEX 値として2桁毎に出力できます。（全１０桁なので、0x10～0x14 の計5個分のポイントアドレスが存在します） たとえば 0x13 のポイントアドレスに 0x8F のバイトデータを送信した場合、7seg の7～8桁目が「8F」と表示されます。 主にマイコンのデバッグ用途等にお使い頂けると思います。

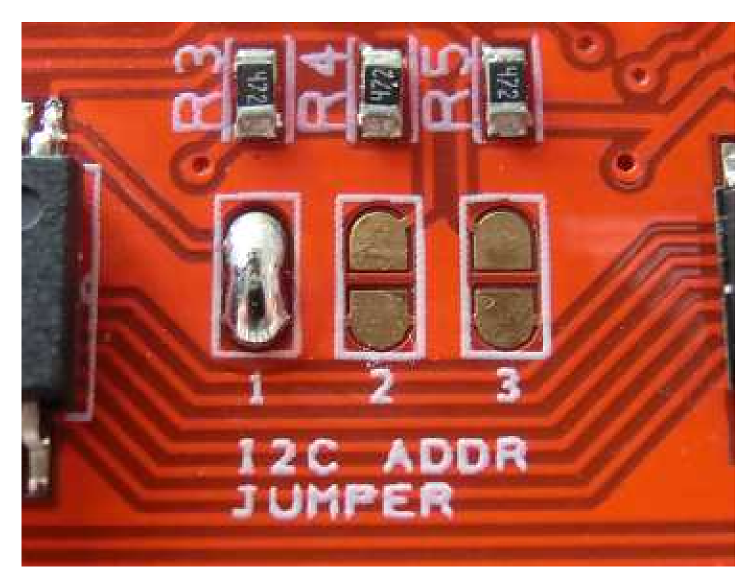
１５．輝度更新指示　（Point Address = 0x20 – 0x29）

当モジュールは、１０桁それぞれ個別に、輝度を１６段階で調節できます。

送信するデータは 0x00（消灯）～0x0f（最大輝度）となります。 それ以外のデータを送信した場合は0x0f を送信したものと見なされます。

１６．I2C カスタムスレーブデバイスアドレス

当モジュールは、0xC0、0xC2、0xC4、0xC6、0xC8、0xCA、0xCC、0xCE の全8種類のプリセット I2C スレーブデバイスアドレス（以下スレーブアドレス）を割り当てることができます。 割り当ては下図のハンダジャンパーを使用します。



※ショート状態を「1」とした場合、以下のような対応になります。

Jumper	000	100	010	110	001	101	011	111
Address	0xC0	0xC2	0xC4	0xC6	0xC8	0xCA	0xCC	0xCE

つまり上図のように1だけをショート状態（100）とした場合は、スレーブアドレスが 0xC2 になります。

更に、上記0xC0～0xCE 以外のスレーブアドレスを割り当てたい場合は、若干手間は掛かりますが、下記の手順で自由に割り当てることができます。

① ハンダジャンパーを１２３全てショート状態にします。
② 電源を接続したら、I2C スレーブアドレス=0xCE に対して 0xF0 のポイントアドレスを送信し、続けて割り当てたいスレーブアドレスを1byte（8bit）送信して下さい。
③ 送信してから2秒後に電源を切断して下さい。（すぐに切断すると記録されません）

以上でカスタムスレーブアドレスの登録は完了です。 登録したスレーブアドレスは内蔵のフラッシュメモリに記憶されますので、明示的に変えない限りは電源を落としても消えません。

尚、ハンダジャンパーは１２３全てショートしたままの状態でご利用下さい。 ハンダジャンパーの状態を変えると、プリセットアドレスが優先されます。

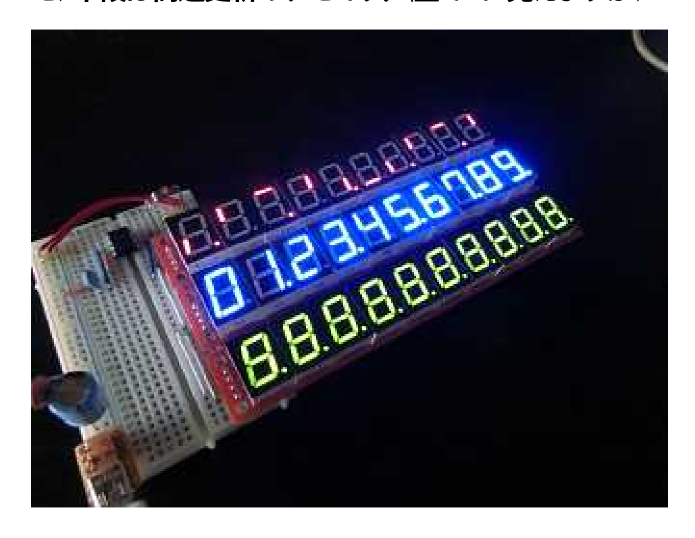
登録したカスタムスレーブアドレスが判らなくなった場合など、カスタムスレーブアドレスをリセットしたい場合は、2番のハンダジャンパーのみショートさせた状態で電源を投入して下さい。電源を投入した瞬間にフラッシュメモリに記録されたカスタムスレーブアドレスが 0xCE にリセットされますので、全ショート（111）時のスレーブアドレスは 0xCE に戻ります。

尚、I2C には、その規格上あらかじめ予約されている（使用できない）スレーブアドレスが存在します。0x00〜0x0F、0xF0〜0xFF が予約されていますので、それ以外のスレーブアドレスをご利用下さい。

また、I2C のスレーブアドレスは競合を避けて下さい。I2C はマスターとスレーブの双方向通信で成り立っていますので、スレーブアドレスが競合していると正しく動作しません――が、競合デバイスを3つ接続して試したところ、問題なく全て同じ動作をしました。スペックが完全同一な為かもしれません。



下図は、20MHz 駆動の PIC12F675（8pinPIC）で、リアルタイムに3つのモジュールを高速表示更新しているデモです。動画で見ないと判りづらいですが、上段はカスタムビット点灯デモ、中段は輝度制御デモ、下段は高速更新のデモです。（全て8に見えますが、0123456789 を高速スクロールしています）



１７．点灯試験モード設定　（PointAddress = 0xF2）

当モジュールは、電源の接続時に全灯点灯試験を行います。 これは当モジュールのプロセッサに異常がないか、また7seg に不具合が起きていないかをチェックする為の動作ですが、この動作を OFF にしたり、別の点灯方法を選択することができます。

点灯試験モードを切り替えるには、I2C で接続する必要があります。
まずは、当モジュールに電源を接続し、点灯試験が終了してから、ポイントアドレス 0xF2 に対して、モードを示す 0～2 の値を送信し、2秒以上待ってから電源を切断して下さい。(必ず2秒以上電源を保って下さい。データの送信後、2秒以上待たないと内蔵のフラッシュメモリに設定値が記憶されません)

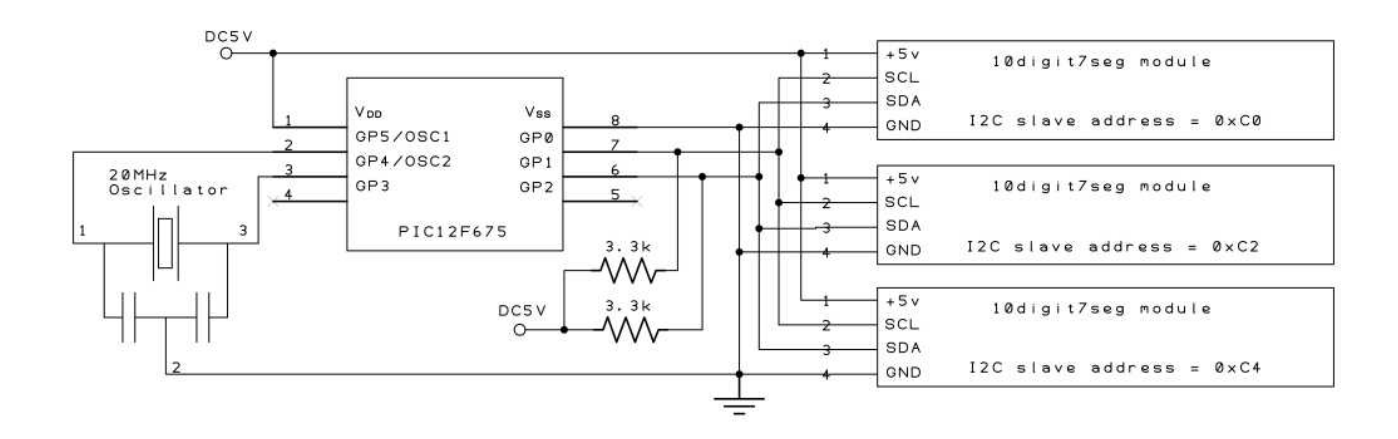
点灯試験モードは以下の3種類があります。
出荷時は「1」が設定されています。

設定値	動作	備考
0	点灯試験 OFF　(※)	
1	フェードイン・フェードアウト動作による点灯試験	※出荷時デフォルト
2	単純な全灯 ON/OFF による点灯試験	

※設定値=0 (点灯試験 OFF) の場合、すぐに I2C の入力データの表示状態に入ります。
※点灯試験中も I2C の入力は受け付けています。 点灯試験中に送信したデータは点灯試験完了後に表示されます。

１８．デモンストレーションのサンプル回路、およびプログラムコード

ここでは、16章の最後の写真と同じデモンストレーションを実行する回路とプログラムコードを掲載します。 I2C の機能を持たない PIC12F675 (8pin) で当モジュール×3個を個別に高速制御する様子を観察することができます。 まずは回路図です。



当モジュール以外に用意するものは、PIC12F675。 20MHz のセラロック。 3.3kΩの抵抗器×2本だけです。 DC5V には安定電源が来ている前提です。

次にプログラムコードですが、Microchip 社の MPLAB IDE (v8.80 以降) と、HI-TECH C Compiler for PIC10/12/16 MCUs (Lite Mode) V9.83 を使用した例を掲載します。 PIC12F675 のプロジェクトを新規作成し、以下のソースコードをそのまま記述すれば OK です。

尚、HI-TECH C 以外にも、mikroElektronika 社の Mikro C などがお薦めできます。標準でハードウェア I2C とソフトウェア I2C の両方に対応していますので簡単に I2C マスター機能を実装できます。

main.c

```
// ================================================================
// 10digit 7seg module - demonstration program for PIC12F675
// ST:2012/04/01 ST:2012/04/01
// Copyright(C)2012- Kitalab / T.Kitakubo
// http://store.kitalab.com/
// ----------------------------------------------------------------------------
// Compiler   : HI-TECH C Compiler for PIC10/12/16 MCUs (Lite Mode)  V9.83
// PIC        : PIC12F675
// Oscillator : HS/20MHz
// Power      : 5V
// ================================================================
#include <pic.h>
#include "pic12f675.h"

// ================================================================
// Configuration
```

```
// ======================================================================
__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_ON & MCLRE_OFF & BOREN_ON & CP_OFF & CPD_OFF);
#define _XTAL_FREQ 20000000      // 20MHz

// ======================================================================
// [macro] I2C setup
// ======================================================================
#define SCL        GP0          // SCL=GP0
#define SDA        GP1          // SDA=GP1
#define SDATRS     TRISIO1

// ======================================================================
// [function] I2C I/F
// ======================================================================
// [I2C] Start condition
void i2c_start(void)
{
    SDA = 1;
    SCL = 1;
    __delay_us(5);
    SDA = 0;
    __delay_us(5);
    SCL = 0;
}
// [I2C] Stop condition
void i2c_stop(void)
{
    SDA = 0;
    SCL = 1;
    __delay_us(30);
    SDA = 1;
    __delay_us(10);
    SCL = 0;
}
// [I2C] Write data
void i2c_write(unsigned char data)
{
    unsigned char i;
    for(i=0; i<8; i++) {
        if(data&0x80) SDA=1;
        else SDA=0;
        data <<= 1;
        SCL = 1;
        __delay_us(1);
        SCL = 0;
        __delay_us(1);
    }
    SDA = 1;
    SDATRS = 1;
    __delay_us(1);
    SCL = 1;
    for(i=0; i<5 && SDA; i++)
        __delay_us(1);
    SCL = 0;
    SDATRS = 0;
}

// ======================================================================
// [function] Main
// ======================================================================
```

```
void update7segDevice(unsigned char slave_address, unsigned char point_address, unsigned char *data)
{
    unsigned char i=0;
    i2c_start();
    i2c_write(slave_address);
    i2c_write(point_address);
    for(i=0; i<10; i++)
        i2c_write(data[i]);
    i2c_stop();
}
void main(void)
{
    // [var] I2C slave device address
    unsigned short nD1=0xC0, nD2=0xC2, nD3=0xC4;
    // [var] Work
    unsigned short i1=0, i2=0, nC1=0, nC2=0, j=0, nWK=0x00;
    unsigned long nUD=0xffff;
    unsigned int nStep=1;
    // [var] Array
    unsigned char nDAT[10];
    unsigned char nLUX[10];
    // [var] Const
    const unsigned char sDigit[] = {
        //.GFEDCBA
        0b00111111,        // 0  A
        0b10000110,        // 1 F B
        0b01011011,        // 2  G
        0b11001111,        // 3 E C
        0b01100110,        // 4  D.
        0b11101101,        // 5
        0b01111101,        // 6
        0b10100111,        // 7
        0b01111111,        // 8
        0b11101111};       // 9

    // Initialize PIC
    CMCON    = 0b00000111;
    ANSEL    = 0b00000000;
    PEIE     = 0;
    TRISIO   = 0b00000000;
    GPIO     = 0b00000000;

    // Waiting for 500ms
    __delay_ms(500);

    // Main loop
    while(1){

        // Pattern change
        nStep--;
        if(nStep==0){
            nStep=1200;

            nWK = nD1;
            nD1 = nD2;
            nD2 = nD3;
            nD3 = nWK;

            for(j=0; j<10; j++)
                nLUX[j] = 0xf;
```

```
            update7segDevice(nD1, 0x20, nLUX);
            update7segDevice(nD2, 0x20, nLUX);

            for(j=0; j<10; j++){
                nDAT[j] = sDigit[j];
                nLUX[j] = 0x5 + j;
            }
            nUD = 0xffff;
            update7segDevice(nD3, 0x00, nDAT);
            update7segDevice(nD3, 0x20, nLUX);
        }

        // Demo.1 (High speed scroll)
        for(j=0; j<10; j++) nDAT[j] = sDigit[(i1+j)%10];
        if(++i1>9) i1=0;
        update7segDevice(nD1, 0x00, nDAT);

        // Demo.2 (Segment control)
        if(nC1>12){
            nC1 = 0;
            for(j=0; j<10; j++){
                switch((i2+j)%6){
                    case 0: nDAT[j]=0b10000001; break;
                    case 1: nDAT[j]=0b00000010; break;
                    case 2: nDAT[j]=0b10000100; break;
                    case 3: nDAT[j]=0b00001000; break;
                    case 4: nDAT[j]=0b10010000; break;
                    case 5: nDAT[j]=0b00100000; break;
                }
            }
            if(++i2>=6) i2=0;
            update7segDevice(nD2, 0x00, nDAT);
        }
        nC1++;

        // Demo.3 (Brightness control)
        if(nC2>14){
            nC2 = 0;
            for(j=0; j<10; j++){
                nWK = (nUD>>j)&0x1;
                if(nWK){
                    if(nLUX[j]==0xF){
                        nLUX[j]--;
                        nUD &= ~(0x1<<j);
                    }else nLUX[j]++;
                }else{
                    if(nLUX[j]==0x0){
                        nLUX[j]++;
                        nUD |= 0x1<<j;
                    }else nLUX[j]--;
                }
            }
            update7segDevice(nD3, 0x20, nLUX);
        }
        nC2++;
    }
}
```

１９．仕様

項目	内容	備考
電源電圧	DC5V±10%	安定電源必須（DC4.5V – 5.5V）
消費電流	5.4mA	モジュール自体の消費電流
通信方式	I2C-BUS / 400kbps / 7bit	
表示方式	１０桁7セグメントＬＥＤ（※）	カソードタイプ
点灯試験	起動時全灯点灯試験あり	OFF、パターン1・2の3種類から選択可能
寸法	139.0mm × 21.0mm	
重量	7g	モジュール基板のみの重量

※7セグメントＬＥＤ、及び制限抵抗は付属しません。 別途ご用意下さい。

２０．免責事項

- 当モジュールは、専用に開発したチェックシステムにより、全てのモジュールに通電を行い、正常動作を確認した後で出荷していますが、不審な点が御座いましたらお問い合わせ下さい。
- 当モジュールは電子工作レベルの知識、更にマイコンプログラミングレベルの知識を持つエンジニアの方のご利用を想定しています。 技術的な質問にはお答え致しかねますので御了承下さい。
- 当モジュールを試用、使用または使用不能により生じた如何なる損害、損失についても、当方は一切の責任を負わず補償致しません。 自己責任でご利用下さい。
- ハンダ付けの工作ミス、適合外の 7seg を使用したことによる表示不良、端子の接続ミス、電源の逆接続等による破損、電源電圧の許容範囲を超える電圧を供給した場合の破損、物理的破損等、加工中、加工後に発生した不良は初期不良にはなりません。
- 本ドキュメントの無断使用、無断転載を禁じます。

２１．開発元・連絡先

KitaLab online store / 北ラボ

store@kitalab.com

http://store.kitalab.com/